(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 November 2001 (01.11.2001)

PCT

(10) International Publication Number
WO 01/82133 A2

(51) International Patent Classification[7]: G06F 17/30

(21) International Application Number: PCT/US01/11829

(22) International Filing Date: 11 April 2001 (11.04.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/547,190 11 April 2000 (11.04.2000) US

(71) Applicant: INFORMATICA CORPORATION [US/US]; 1200 Chrysler Drive, Menlo Park, CA 94025 (US).

(72) Inventor: GOVINDARAJ, Naresh, K.; 6495 Trinidad Drive, San Jose, CA 95120 (US).

(74) Agents: GALLENSON, Mavis, S. et al.; Ladas & Parry, 5670 Wilshire Boulevard, Suite 2100, Los Angeles, CA 90036-5679 (US).

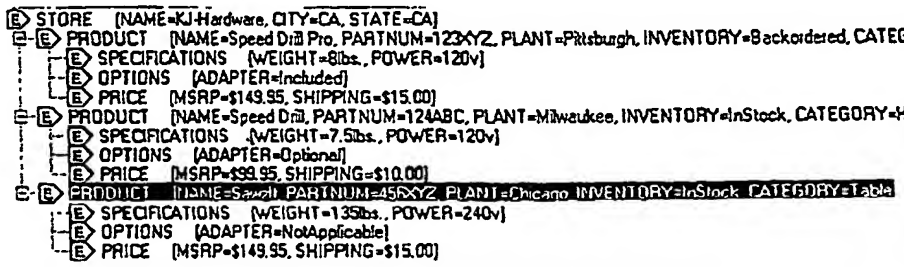
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: XML FLATTENER



(57) Abstract: A method and apparatus for automatically flattening an XML file. XML files are stored in a client or server. The user specifies one of the XML files or a particuliar subset of the selected XML file to flatten. The user also specifies which elements and/or attributes of the selected subset of the XML file is of interest. The elements and the attributes of interest to the user for the selected subset are then parsed by a parser process. The parsed elements and attributes are then automatically arranged into a flat format having rows and columns as defined by the user.

XML FLATTENER

FIELD OF THE INVENTION

The present invention relates to a method for flattening an XML file.

BACKGROUND OF THE INVENTION

The Internet is a general purpose, public, global computer network which allows computers hooked into the Internet to communicate and exchange digital data with other computers also on the Internet. Once a computer is coupled to the Internet, a wide variety of options become available. Some of the myriad functions possible over the Internet include sending and receiving electronic mail (e-mail) messages, logging into and participating in live discussions, playing games in real-time, viewing pictures, watching streaming video, listening to music, going shopping on-line, downloading and/or uploading files, and browsing different web sites, etc.

Most of these functions are made available to the casual internet user through the use of browsers. The browser facilitates the communication with an internet site through a given protocol. The orginal protocol developed to handle data transmissions between a user's client computer and the server computer hosting the web site is known as Hypertext Transfer Protocol (HTTP). This protocol specifies a set of technical rules by which client and server programs can communicate with one another. In this manner, HTTP is used to transfer data

between servers and clients via a browser program (e.g., Navigator or Explorer) over a part of the Internet known as the World Wide Web or "the Web." HTTP enables a user to simply place a cursor on a displayed hypertext link and click on it. This automatically takes the user to the appropriate web page, to other desired information, or to another resource located on the same or different server on the Internet.

Although HTTP was widely adopted as the defacto protocol for navigating the internet, it soon became outdated. A more versatile protocol known as Extensible Markup Language (XML) is fast gaining popularity amongst web designers. The XML specification originates from the World-Wide Web consortium(W3C) and is platform, application and vendor independent. Basically, XML is a markup language for documents containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). XML provides a data standard that can encode the content and semantics of a document. Almost all documents have some structure. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents. The word "document" refers not only to traditional documents, but also to the miriad of other XML "data formats". These include vector graphics, e-commerce transactions, mathematical equations, object meta-data, server APIs, and a thousand other kinds of structured information.

In all cases, XML requires a hierarchical programming format. To understand the components of an XML document, it is useful to look at a example. The following is a sample XML file--store.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE STORE SYSTEM "store.dtd">
<STORE NAME="K&J Hardware" CITY="CA" STATE="CA">
<PRODUCT NAME="Speed Drill Pro" PARTNUM="123XYZ"
PLANT="Pittsburgh" INVENTORY="Backordered" CATEGORY="Shop-
Professional">
        <SPECIFICATIONS WEIGHT="8lbs." POWER="120v"/>
        <OPTIONS ADAPTER="Included" CASE="HardShell"/>
        <PRICE MSRP="$149.95" WHOLESALE="$99.95"
STREET="$129.95" SHIPPING="$15.00"/>
        <NOTES>Professional Version of the top selling "Speed
Drill" from the consumer line.</NOTES>
</PRODUCT>
<PRODUCT NAME="Speed Drill" PARTNUM="124ABC"
PLANT="Milwaukee" INVENTORY="InStock" CATEGORY="HandTool">
        <SPECIFICATIONS WEIGHT="7.5lbs." POWER="120v"/>
        <OPTIONS ADAPTER="Optional" CASE="Soft"
FINISH="Polished"/>
        <PRICE MSRP="$99.95" WHOLESALE="$69.95"
STREET="$79.95" SHIPPING="$10.00"/>
```

```
        </PRODUCT>
        <PRODUCT NAME="SawzIt" PARTNUM="456XYZ" PLANT="Chicago"
    INVENTORY="InStock" CATEGORY="Table">
                        <SPECIFICATIONS WEIGHT="135lbs." POWER="240v"/>
                        <OPTIONS ADAPTER="NotApplicable"
    CASE="NotApplicable" FINISH="Metal"/>
                        <PRICE MSRP="$149.95" WHOLESALE="$99.95"
    STREET="$129.95" SHIPPING="$15.00"/>
        </PRODUCT>
        </STORE>
```

The above XML sample represents products sold in a store. The first line of the XML file indicates the XML specification version:

```
        <?xml version="1.0" encoding="UTF-8"?>
```

If the XML file has an associated DTD file then this is specified as indicated in the second line of the above example:

```
        <!DOCTYPE STORE SYSTEM "store.dtd">
```

The XML document consists of a hierarchy of elements. Each element begins with a start tag and ends with an end tag. The root element is the topmost element. In the above example, the root element is STORE. The start and end tags for the STORE element is <STORE> and </STORE> respectively. The

PRODUCT element is a sub element that appears under the STORE element. In the above example, there are three products in the K&J Hardware store. An element can have attributes. In the above example, the PRODUCT element has the following attributes: NAME, PARTNUM, PLANT, and INVENTORY. The hierarchical structure of the store.xml document given above can be represented in a "tree" format as shown in Figure 1. It can be seen that the top hierarchy consists of a store (KJ-Hardware). The store, in the next hierarchical level, has three products (Speed Drill Pro, Speed Drill, and Sawzlt). The next hierarchical level consists of the Specifications, Options, and Price for each of the products.

Although this hierarchical format lends itself quite handily for designing web pages, it is ill-suited for other types of applications. Unfortunately, many software programs require a "flat" type of data structure having rows and columns of data. For example, data warehousing, data mining, and data mart applications all typically require a "flat" type of data structure to operate from. Currently, companies are taking a programmatic approach to converting XML data to flat data. However, this is custom tailoring approach is quite labor intensive, time consuming, and expensive.

Thus, there exists a need in the prior art for an apparatus and method for automatically flattening any XML file. The present invention provides a unique, novel solution to this problem.

SUMMARY OF THE INVENTION

The present invention pertains to a method and apparatus for flattening an XML file. Basically, XML files are stored in a client or server. The user specifies one of the XML files or a particular subset of the selected XML file to flatten. The user also specifies which elements and/or attributes of the selected subset of the XML file is of interest. The elements and attributes of interest to the user for the selected subset are then parsed by a parser process. The parsed elements and attributes are then automatically arranged into a flat format having rows and columns as defined by the user.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

Figure 1 shows a hierarchical structure of the store.xml document.

Figure 2 shows an XML Source Metatdata Analysis process.

Figure 3 shows an XML Source Flattening process.

Figure 4 shows three processes: XML Parser, XML Flattener, and XML Views.

Figure 5 shows an exemplary computer system upon which the present invention may be practiced.

DETAILED DESCRIPTION

An apparatus and method for flattening an XML file is described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be obvious, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid obscuring the present invention.

Referring to Figure 2, an XML Source Metatdata Analysis process is shown. Initially, an XML source 201 is accessed by an XML View process 202. The XML View process 202 operates on a user specified subset of an XML document retrieved from the XML source 201. This subset corresponds to a set of logically related elements and attributes. The XML View process 202 specifies how and what to flatten in an XML source file. The XML View process 202 is defined by the user in a control file. It should be noted that an XML source 201 can have multiple XML View processes 202. The data is then transformed according to the Metadata XML 203. The result is then stored in a flat file source 204.

Referring to Figure 3, an XML Source Flattening process is shown. The XML source 301 is operated upon by the XML View process 302 (e.g., a control file) and Metadata XML 303. The converted flat file 304 can then be processed by a Reader process 305.

An example of an XML Views for a particular Store is shown below.

XML Source (Store.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE STORE SYSTEM "store.dtd">
<STORE NAME="KJ-Hardware" CITY="SF" STATE="CA">
<TEL Type="Direct" TelNum="408-111-2222"/>
<TEL Type="FAX" TelNum="408-333-2222"/>
<PRODUCT NAME="Speed Drill Pro" PARTNUM="123XYZ"
PLANT="Pittsburgh" INVENTORY="Backordered" CATEGORY="Shop-
Professional">
        <SPECIFICATIONS WEIGHT="8lbs." POWER="120v"/>
        <OPTIONS ADAPTER="Included" />
        <PRICE MSRP="$149.95" SHIPPING="$15.00"/>
</PRODUCT>
<PRODUCT NAME="Speed Drill" PARTNUM="124ABC" PLANT="Milwaukee"
INVENTORY="InStock" CATEGORY="HandTool">
        <SPECIFICATIONS WEIGHT="7.5lbs." POWER="120v"/>
        <OPTIONS ADAPTER="Optional"/>
        <PRICE MSRP="$99.95"  SHIPPING="$10.00"/>
</PRODUCT>
<PRODUCT NAME="SawzIt" PARTNUM="456XYZ" PLANT="Chicago"
INVENTORY="InStock" CATEGORY="Table">
        <SPECIFICATIONS WEIGHT="135lbs." POWER="240v"/>
```

```
            <OPTIONS ADAPTER="NotApplicable"/>
            <PRICE MSRP="$149.95" SHIPPING="$15.00"/>
    </PRODUCT>
    <MANAGER>James Bond</MANAGER>
    </STORE>
```

An example of an XML Views corresponding to the Inventory of the above Store is given below.

XML Source (INVENTORY)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMLFLAT SYSTEM "xmlflat.dtd">
<XMLFLAT>
  <XMLVIEW NAME="INVENTORY" DELIMITER =
"|">
    <ELEM NAME="PRODUCT" PROCESSTYPE="KEEP
>
      <ATTR NAME="PLANT"
PROCESSTYPE="IGNORE"/>
      <ATTR NAME="INVENTORY"
PROCESSTYPE="IGNORE"/>
      <ATTR NAME="CATEGORY"
OPTIONAL="YES"/>
    </ELEM>
```

```
    <ELEM NAME="OPTIONS"
PROCESSTYPE="IGNORE"/>
    <ELEM NAME="SPECIFICATIONS OPTIONAL=
YES >
      <ATTR NAME="POWER"
PROCESSTYPE="IGNORE"/>
    </ELEM>
   <ELEM NAME= TEL" PROCESSTYPE="IGNORE"/>
   <ELEM NAME= MANAGER"
PROCESSTYPE="IGNORE"/>
   </XMLVIEW>
</XMLFLAT>
```

The flat file corresponding to the examples given above is now shown below.

Flat File

STORE_NAME;STORE_CITY;STORE_STATE;PRODUCT_NAME;PRODUCT_PARTNUM;PRODUCT_CATEGORY;SPECIFICATIONS_WEIGHT;PRICE_MSRP;PRICE_SHIPPING

KJ-Hardware;SF;CA;Speed Drill Pro;123XYZ;Shop-Professional;8lbs.;$149.95;$15.00

KJ-Hardware;SF;CA;Speed Drill;124ABC;HandTool;7.5lbs.;$99.95;$10.00

KJ-Hardware;SF;CA;SawzIt;456XYZ;Table;135lbs.;$149.95;$15.00

The control file for the examples given above is given as follows. It should be noted that the control file exists in an XML format, contains the XML view definitions, and defaults to xmlfctrl.xml.

Control File DTD

```
<!ELEMENT XMLFLAT (XMLVIEW+)>
<!ELEMENT XMLVIEW (ELEM*)>
<!ATTLIST XMLVIEW NAME CDATA #IMPLIED
    DELIMITER CDATA #IMPLIED
    SRCNAME CDATA #IMPLIED
    SRCDESC CDATA #IMPLIED
    ELEMPROCESSTYPE (KEEP | IGNORE ) "KEEP" >
<!ELEMENT ELEM (ATTR*)>
<!ATTLIST ELEM NAME CDATA #IMPLIED
    PROCESSTYPE (KEEP | IGNORE ) "KEEP"
    OPTIONAL (YES|NO) "NO"
    MULTCOL CDATA #IMPLIED
    ATTRPROCESSTYPE (KEEP | IGNORE ) "KEEP" >
<!ELEMENT ATTR (#PCDATA)>
<!ATTLIST ATTR NAME CDATA #IMPLIED
    PROCESSTYPE (KEEP | IGNORE) "KEEP"
    OPTIONAL (YES|NO) "NO" >
```

An XML Views for a multicolumn applications is now shown below.

XML View (INVENTORY2)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMLFLAT SYSTEM "xmlflat.dtd">
<XMLFLAT>
  <XMLVIEW NAME="INVENTORY2" DELIMITER = ";">
   <ELEM NAME="PRODUCT" PROCESSTYPE="KEEP" OPTIONAL="YES">
    <ATTR NAME="PLANT" PROCESSTYPE="IGNORE"/>
    <ATTR NAME="INVENTORY" PROCESSTYPE="IGNORE"/>
    <ATTR NAME="CATEGORY" OPTIONAL="YES"/>
   </ELEM>
   <ELEM NAME="OPTIONS" PROCESSTYPE="IGNORE"/>
   <ELEM NAME="SPECIFICATIONS">
    <ATTR NAME="POWER" PROCESSTYPE="IGNORE"/>
   </ELEM>
   <ELEM NAME="TEL" MULTCOL = "2"/>
  </XMLVIEW>
</XMLFLAT>
```

Its corresponding flat file is given as follows.

Multicolumn Flat File

```
STORE_NAME;STORE_CITY;STORE_STATE;TEL_Type;TEL_Type1;TEL_TelNum;
TEL_TelNum1;PRODUCT_NAME;PRODUCT_PARTNUM;PRODUCT_CATEGOR
Y;SPECIFICATIONS_WEIGHT;PRICE_MSRP;PRICE_SHIPPING;MANAGER
```

KJ-Hardware;SJ;CA;Direct;FAX;408-111-2222;408-333-2222;Speed Drill Pro;123XYZ;Shop-Professional;8lbs.;$149.95;$15.00;James Bond

KJ-Hardware;SJ;CA;Direct;FAX;408-111-2222;408-333-2222;Speed Dril XL;124ABC;HandTool;7.5lbs.;$99.95;$10.00;James Bond

KJ-Hardware;SJ;CA;Direct;FAX;408-111-2222;408-333-2222;Chain Saw Pro;456XYZ;Table;135lbs.;$149.95;$15.00;James Bond

KJ-Hardware;SJ;CA;Direct;FAX;408-111-2222;408-333-2222;Socket Set;123XYZ;Table;135lbs.;$149.95;$15.00;James Bond

In order to invoke a flattening process, an XMLFLAT invocation might look like: XMLFLAT [options] xml_source_file [flat_dest_file]. The following are the options that can be specified:

-c< control file >	Control file/view for processing XML source file
-v< view_name>	View name in control file
-a	Append rows to flattened file
-m	source XML metadata analysis
-p<repository connect parameters>	Used for Source XML metadata push to repository
-ps<repository connect parameters>	Same as -p option, but will be prompted for password info.

Some examples of invocations include:

XML Source Analysis:

XMLFlat -m -v INVENTORY c:\data\store.xml c:\data\out.xml

XML Source Metedata push

XMLFlat -v INVENTORY

-p rep1 administrator db1 naresh pwrep pwdb folderx 1.01
c:\data\store.xml c:\data\out.xml

XML Flattening

XMLFlat -v INVENTORY c:\data\store.xml c:\data\out.xm

In the currently preferred embodiment, as shown in Figure 4, there are three processes which are used: XML Parser 401, XML Flattener 402, and XML Views 403. Each of these processes are now described in detail.

There are XML parsers available from different vendors. There are validating and non validating parsers. The validating parsers also verify the DTD conformance of the XML file. The following are some of the XML parsers available: IBM's XML4j; IBM's XML4c; Microsoft's MSXML; Oracle's V2 parser; Sun's "Java Project X"; and DataChannel XML Parser for Java. There are two major types of XML (or SGML) APIs: 1) tree-based APIs; (DOM) and 2) event-based APIs (SAX). A tree-based API compiles an XML document into an internal tree structure, then allows an application to navigate that tree. The Document Object Model (DOM) working group at the World-Wide Web consortium is developing a standard tree-based API for XML and HTML documents.

An event-based API, on the other hand, reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface.

Tree-based APIs are useful for a wide range of applications, but they often put a great strain on system resources, especially if the document is large (under very controlled circumstances, it is possible to construct the tree in a lazy fashion to avoid some of this problem). Furthermore, some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

In both of these cases, an event-based API provides a simpler, lower-level access to an XML document: one can parse documents much larger than the available system memory, and one can construct their own data structures using their callback event handlers.

The XML Flattener is an application which takes an XML file as input and produces a flat file that contains the XML element values and attributes in rows. Once the flat file is produced it can be used as a data source and the data can be read in by the reader. The invocation of the XML flattener can occur as a pre-session command by the server. In the currently preferred embodiment, the XML Flattener uses a imbedded XML parser to parse the XML file. As the elements in the XML hierarchy are parsed, they are collected in buffers and then written to a file with a user specified delimiter. Once the entire XML file is processed then the out flat file will be closed, and will be ready to be read by the XML reader. The flat file will have header information followed by data rows. The header will contain the column names. The column names can be generated

by concatenating element names in the hierarchy chain. The following is an sample flattened representation of the sample store.xml file.

STORE_NAME; STORE_STATE; PRODUCT_NAME; PRODUCT_PLANT; PRODUCT_PARTNUM; PRODUCT_CATEGORY; PRODUCT_INVENTORY; SPECIFICATIONS_POWER; SPECIFICATIONS_WEIGHT; OPTIONS_ADAPTER; PRICE_MSRP; PRICE_SHIPPING
KJ-Hardware; CA; Speed Drill Pro; Pittsburgh; 123XYZ; Shop-Professional; Backordered;120v; 8lbs.; Included; $149.95; $15.00
KJ-Hardware; CA; Speed Drill; Milwaukee; 124ABC; HandTool; InStock;120v; 7.5lbs.; Optional; $99.95; $10.00
KJ-Hardware; CA; SawzIt; Chicago; 456XYZ; Table; InStock; 240v; 135lbs.; NotApplicable; $149.95; $15.00

The first row(spread through 3 rows in the document) contains the column names. There are three data rows that has the element and attribute values for the three products in the store. The semicolon is used as the delimiter in the above flat file. The delimiter itself should be user specified parameter to the XML flattener.

Filters are used in case the user is only interested in a subset of the elements/attributes. The user should be able to specify the elements/attributes that are to be parsed or those to be ignored. The following is a flat file that has been filtered to exclude the SPECIFICATION element:

STORE_NAME; STORE_STATE; PRODUCT_NAME; PRODUCT_PLANT; PRODUCT_PARTNUM; PRODUCT_CATEGORY; PRODUCT_INVENTORY; OPTIONS_ADAPTER; PRICE_MSRP; PRICE_SHIPPING

KJ-Hardware; CA; Speed Drill Pro; Pittsburgh; 123XYZ; Shop-Professional; Backordered; Included; $149.95; $15.00

KJ-Hardware; CA; Speed Drill; Milwaukee; 124ABC; HandTool; InStock; Optional; $99.95; $10.00

KJ-Hardware; CA; SawzIt; Chicago; 456XYZ; Table; InStock; NotApplicable; $149.95; $15.00

Element filtering can be used for example to filter out the routing related tags from BizTalk XML schemas. Filters by Value provide a way to filter information from the XML file based on values of an element or attribute. For example, the user may want be only interested in all the stores in California. Having filters will reduce the size of the flattened XML file, thereby reducing the processing time for the reader to read the file.

Single versus repeatable elements are handled as follows. The first line in store.dtd contains <!ELEMENT STORE (PRODUCT+)>. The '+' after the PRODUCT declaration indicates that the PRODUCT sub element can occur more than once under the STORE element. Hence , there is a 1 to many relationship between STORE and PRODUCT. What if the STORE element had more than one sub element that can have multiple occurrences, such as the following:

<!ELEMENT STORE (PRODUCT+ EMPLOYEE+)>.

The EMPLOYEE sub element can also occur more than one under the STORE element. The XML flattener can process either the PRODUCT or the EMPLOYEE sub element, and not both into the same flat file. The reason is that the PRODUCT and EMPLOYEE entities do not relate to each other. They only relate to the parent STORE. On the other hand, consider the following:

<!ELEMENT STORE (PRODUCT+ EMPLOYEE+ LOCATION)>.

A new sub element LOCATION has been added which has attributes regarding the location of the store. The PRODUCT and LOCATION entities can be used in conjunction, and so can the EMPLOYEE and LOCATION entities. In the case where an element can have more than one repeatable sub element, the XML flattener can process only one of them. In such cases the user would need to indicate which one is to be processed. In the above example, the user can include either PRODUCT or EMPLOYEE to be processed by the XML flattener, and not both at the same time.

The XML Flattener is an executable that can be invoked from the command line. The following parameters can be specified:

-XML file (The source XML file)

-Flat file (The flattened output file)

-Delimiter

-Filter parameters (For Element/Attribute/Value based filtering)

It should be possible to specify the filter parameters so that they represent elements/attributes to be ignored or processed. This will provide more flexibility for the user. The complexity of the input parameters may make it difficult to be processed as command line parameters. If this is the case we may have to provide a way of specifying the parameters in a control file. This control file itself could be an XML file. The details and specification of the parameters and the invocation of the XML flattener is discussed below.

The XML flattener, XMLFLAT, is a command line application that servers two purposes. It is invoked initially for any new XML source file type to capture the flattened XML source metadata and update the repository. It is also invoked as a pre-session command to flatten an XML source file so that the flattened file can be read by the Reader. XMLFlat on completion will return a status code. The value of the status code will indicate if the the XML file was processed successfully or not. The server needs to verify this status code before it starts the session to read the flattened XML file.

The XML flattener can be invoked as follows, from the command line:

XMLFLAT [options] -v viewname xml_source_file [flat_dest_file]

The following are the options that can be specified:

-c< control file >	Control file/view for processing XML source file
-v< view_name>	View name in control file. This is

-a	Append rows to flattened file
-b<batch_file>	Batch more than XML source for flattening
-m	Source XML metadata analysis
-p<repository connect parameters>	Used for Source XML metadata push to repository
-ps<repository connect parameters>	Same as -p option, but will be prompted for password info.

<repository connect parameters>(p) = <repname repuser dbname dbuser rep_pwd db_pwd foldername folderversion>

<repository connect parameters>(ps) = <repname repuser dbname dbuser foldername folder version>

The xml_source_file is the full path of the source XML file. The source XML file is needed for XML source analysis and for flattening. The flat_dest_file is the output file for the flattened rows. The output file is required for the flattening option. The flat_dest_file is not needed with the -m, -p, and -ps options. The -v option is used to specify the view name to be used for analysis or flattening. This is a required option for all XMLFLAT invocations. The -c option allows you to specify a control file other than the default control file. The default control file is xmlfctrl.xml in the same directory as the XMLFLAT executable.

For XML Source Analysis and Metadata Push Options, the -m option is used to analyze the source XML file and capture the source metadata. This

metadata information defines the flat file source and column attributes and the mapping between the XML elements and attributes to flat file columns. This includes flat file source name , business name and delimiter. The column attributes include the column name, business name, datatype (STRING always), precision. The metadata is stored in XML format in the metadata.xml file. This is used to push the flat file source metadata into the Informatica repository. This is also used during the source flattening process.

The -p option is used to push the XML source metadata into the repository. This option assumes that the -m option was used earlier to capture the metadata into the metadata.xml file, or it is used in conjunction with the -m option. This option is discussed further in the 'XML Source Metadata Capture' section. The -ps option works the same as the -p option, except that the user will be prompted for the repository and database passwords. The following are example invocations of XMLFLAT for XML source analysis and metadata push:

XML Source Analysis:(-m option)

XMLFLAT -m -v INVENTORY store.xml

XML Source Push (-p option)

XMLFLAT -v INVENTORY -p testrep Administrator Administrator naresh naresh sql_srvr folder1 1.0.0 store1.xml out.txt

XML Source Analysis + Push (-m,-p option)

XMLFLAT -v INVENTORY -m -p testrep Administrator Administrator naresh naresh sql_srvr folder1 1.0.0 store1.xml out.txt

For XML Source Flattening Options, there is no specific option to invoke flattening of XML sources. If the -m,-p, or -ps options are not used, it is assumed that the source XML file is to be flattened. The -a option is used to append rows to the output XML file. If this option is not used then the output XML file will be overwritten. The -b option enables one to specify a batch of source XML files for flattening. With the -b option, a batch file containing a list of source XML files is to be specified. The following is an example batch file. Note that the source XML files appear on separate lines in the batch file.

```
c:\data\tran11.xml
c:\data\tran12.xml
c:\data\tran13.xml
```

The following are example invocations of XMLFLAT for XML source flattening:

```
XMLFLAT -v INVENTORY store.xml out.txt
XMLFLAT -v -a INVENTORY store.xml out.txt  (append option)
XMLFLAT -b batch.txt -v INVENTORY out.txt  (batch option)
```

With respect to XMLFLAT Control File and XML Views, the XML View provides a way for the user to specify a subset of a source XML file, and information regarding how to represent information in a XML document in a row structure. It also contains information about the flat file source. A single

source XML file can have multiple views. An XML view contains the following information:

The view name

Flat file delimiter

Flat file source name and description

Elements and attributes in XML source to be kept or ignored during flattening.

There are optional elements and attributes. For XMLFALT Control File, the following example shows the DTD for the XML Flattener control file. The XML format is seen here as an appropriate format to represent the control file since it is platform independent and we can use a XML parser to parse the control file. It will also allow users to use any of the available XML graphical editors to specify the control parameters.

```
<!ELEMENT XMLFLAT (XMLVIEW+)>
<!ELEMENT XMLVIEW (ELEM*)>
<!ATTLIST XMLVIEW NAME CDATA #REQUIRED
    DELIMITER CDATA #REQUIRED
    SRCNAME CDATA #REQUIRED
    SRCDESC CDATA #REQUIRED
    ELEMPROCESSTYPE (KEEP | IGNORE ) "KEEP" >
<!ELEMENT ELEM (ATTR*)>
<!ATTLIST ELEM NAME CDATA #REQUIRED
    PROCESSTYPE (KEEP | IGNORE ) "KEEP"
```

```
        OPTIONAL (YES | NO) "NO"
        MULTCOL CDATA #IMPLIED
        ATTRPROCESSTYPE (KEEP | IGNORE ) "KEEP" >
    <!ELEMENT ATTR (#PCDATA)>
    <!ATTLIST ATTR NAME CDATA #REQUIRED
        PROCESSTYPE (KEEP | IGNORE) "KEEP"
        OPTIONAL (YES | NO) "NO" >
```

The root element in the above DTD is XMLFLAT. The control file can contain the parameters for several XML source files. The XMLVIEW element, which is a sub-element of XMLFLAT, contains the view specification for an XML source file. The above method allows the same source XML file in different ways. This is useful when the same XML file is to be used as a source for different information. For example, the products information from the store.xml input can be flattened by specifying:

```
XMLFLAT -v  PRODUCT store.xml store_prod.out
```

If the only employee information is to be flattened then one can specify:

```
XMLFLAT -v  EMPLOYEE  store.xml store_emp.out
```

The delimiter to be used in the flattened file is specified by the DELIMITER attribute of the XMLVIEW element. The SRCNAME and SRCDESC attributes of XMLVIEW are the flat file source name and description to be used when we

push the metadata into the Informatica repository. The ELEMPROCESSTYPE attribute of XMLVIEW indicates if elements are to ignored or kept by default. This only applies to elements that do not have a corresponding ELEM sub element specification. The ELEM sub element is discussed later. The default value for ELEMPROCESSTYPE is 'KEEP'.

The XMLVIEW element can have one or more sub elements of type ELEM. The ELEM sub element provides a way to specify how an element/attribute in source XML file is to be processed. The PROCESSTYPE attribute of ELEM has the following possible values:

"KEEP" :	Include this element in the flattened file
"IGNORE" :	Ignore this element, and any sub elements while processing. They will not appear in the flattened file.

The MULTCOL attribute of ELEM specifies that multiple occurrences of the element will be flattened to the same row in the flattened file. The value of MULTCOL specifies the number of times the element can appear. For example, MULTCOL="2" specifies that the element can appear two times in the row.

The OPTIONAL attribute of ELEM indicates if an element is optional. If the element does not exist in the source XML file then a empty string will assigned to it in the flattened file. The default value for OPTIONAL is "NO".

The ATTRPROCESSTYPE attribute of XMLVIEW indicates if attributes are to ignored or kept by default for an element. This only applies to attributes that do not have a corresponding ATTR sub element specification. The ATTR sub element is discussed later. The default value for ATTRPROCESSTYPE is 'KEEP'.

The ATTR sub element of ELEM element provides a way to specify how a specific attribute of an element is to be processed. The PROCESSTYPE attribute of ATTR has the following possible values:

"KEEP": Include this attribute in the flattened file.

"IGNORE": Ignore this attribute while processing. They will not appear in the flattened file.

The OPTIONAL attribute of ATTR indicates if an attribute is optional. If the element does not exist in the source XML file then a empty string will assigned to it in the flattened file. The default value for OPTIONAL is "NO".

The following is an example control file for the store.xml input file. The filter specified will only process the stores in CA.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMLFLAT SYSTEM "xmlflat.dtd">
<XMLFLAT>
 <XMLVIEW NAME="INVENTORY" DELIMITER = "|">
```

```
<ELEM NAME="STORE">
 <ELEMFILTER ATTRNAME="STATE" ATTRVALUE="CA"
OPERATOR = "="/>
</ELEM>
<ELEM NAME="PRODUCT" PROCESSTYPE="ROWTERM"/>
<ELEM NAME="OPTIONS" PROCESSTYPE="IGNORE"/>
<ELEM NAME="SPECIFICATIONS">
 <ATTR NAME="POWER" PROCESSTYPE="IGNORE"/>
</XMLVIEW>
</XMLFLAT>
```

The ELEMFILTER sub element of ELEM provides a way to filter information from the XML source file based on element attribute values. The flattened file will contain rows that satisfy the filter criteria. The filter specification as stated allows filtering based on an attribute value of an element. A composite filter cannot be specified now. The following are the attributes of ELEMFILTER:

ATTRNAME: name of attribute
ATTRVALUE : The filter value
OPERATOR: "EQ" | "LT" | "LE" | "NE" | "GT" | "GE" | "LIKE"

For XML Source Metadata Capture (XML Source Analysis), the metadata describing the flattened XML file consists of the following: the source name describing the XML source and the source field level metadata which includes

the field names, data types and maximum length. Since the XML source analysis is not available in the Designer we need to provide a way for the XMLFLAT application to push the above XML source related metadata into the repository. This can be accomplished by using the -p option of the XMLFLAT:

-p<repository connect parameters>

The user would need the invoke XMLFLAT with the -p option manually to get the XML source metatdata into the repository. From then on when XMLFLAT is invoked via a pre session command, the -p option is not needed. The -m option of XMLFLAT should be used to capture the source XML metadata before the -p option is used, or in conjunction with the -p option. The metadata is captured into an XML format in the metadata.xml file.

The repository connection parameters that is specified with the -p option will be used by XMLFLAT to connect to the repository and push the metadata. For the purpose of this release the data types of the XML source fields will all be the string data type. The maximum length for the fields will be determined by processing the input XML source file and capturing the maximum length of data values.

This way of determining the maximum length may cause a problem. The XML file that is used during the source analysis may have data values that have shorter lengths than the XML files that are used in the sessions. If this happens then the reader will not be able to read the flattened XML file. In this case the

user can invoke the Designer and specify a new maximum field length. The other options are to do the source analysis step again with an update intent. The other way, is for the user to specify maximum lengths(via the control file) to be applied for the fields.

It should be noted that errors can occur during processing of XMLFLAT. Some of the errors can be detected and XMLFLAT will return an error code. In some cases the error cannot be detected and will result in invalid information in the flattened file.

Errors that can be easily detected are, for example:

Missing XML file/control file,

XML file is not "well formed"

XML file is invalid(does not comply to DTD).

Invalid specification of the delimiter

The following situations are detected and handled properly.

Missing elements/attributes in the XML source file that existed during XML source analysis.

New elements/attributes in XML file that did not exist during XML source analysis.

Element tags in different order.

XSLT (XSL Transformation Language) is a language used to "transform" (or reconstruct the structure of) the data structures contained within XML documents. XSLT can be used to transform XML documents into other XML

documents. It could also be used to combine different XML documents into one XML document. XSLT can be used as preprocess step for the XML flattener when there are more than one XML source files which are to be merged before they can be flattened. There are some tools available in the market which lets you build the XSLT visually for such purposes. XSLT can also be used to transform complex XML documents to simpler structures that can be easily flattened. It can also be used to filter XML documents based on certain criteria before they can be processed by the XML flattener.

Figure 5 shows an exemplary computer system upon which the present invention may be practiced. It is appreciated that the computer system 501 of Figure 5 is exemplary only and that the present invention can operate within a number of different computer systems. Computer system 501 of Figure 5 includes an address/data bus 506 for conveying digital information between the various components, a central processor unit (CPU) 502 for processing the digital information and instructions, a main memory 504 comprised of random access memory (RAM) for storing the digital information and instructions, a read only memory (ROM) 503 for storing information and instructions of a more permanent nature. In addition, computer system 501 may also include a data storage device 505 (e.g., a magnetic, optical, floppy, or tape drive) for storing vast amounts of data, and an I/O interface 510 for interfacing with peripheral devices (e.g., computer network, modem, etc.). It should be noted that the client program for performing XML flattening can be stored either in main memory 504, data storage device 505, or in an external storage device. Devices which may be coupled to computer system 501 include a display device 507 for displaying

information to a computer user, an alphanumeric input device 508 (e.g., a keyboard), and a cursor control device 509 (e.g., mouse, trackball, light pen, etc.) for inputting data and selections.

Thus, an apparatus and method for automatically flattening any XML file is described. The foregoing descriptions of specific embodiments of the present invention have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, and obviously many modifications and variations are possible in light of the above teaching. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the Claims appended hereto and their equivalents.

CLAIMS

What is claimed is:

1. An apparatus for flattening an XML file, comprising:

means for storing a plurality of XML files;

means for specifying a particular one of said XML files;

means for specifying a particular subset of the particular file, wherein the subset comprises logically related elements and attributes;

means for parsing the elements and attributes of the particular subset of the particular file;

means for arranging parsed elements and attributes of the particular subset in a flat format having rows and columns.

2. A method for flattening an XML file, comprising the steps of:

storing a plurality of XML files;

specifying a particular one of said XML files;

specifying a particular subset of the particular file, wherein the subset conprises logically related elements and attributes;

parsing the elements and attributes of the particular subset of the particular file;

arranging parsed elements and attributes of the particular subset in a flat format having rows and columns.

3. The method of Claim 1 further comprising the step of using the flat format in a data mart application.

4. The method of Claim 2 further comprising the step of filtering the elements and attributes, wherein only filtered elements and attributes are parsed.

5. The method of Claim 2 further comprising the step of converting the flat file back to XML.

6. The method of Claim 2 further comprising the step of specifying a view name, a flat file delimiter, a flat file source name and description, elements and attributes in XML source, optional elements and attributes, and options to process multiple occurrence elements.

7. The method of Claim 2, wherein the method is platform independent.

8. The method of Claim 2 further comprising the step of performing metadata source analysis.

9. The method of Claim 2 further comprising the step of perfmorming XML source metadata capture.

STORE [NAME=KJ-Hardware, CITY=CA, STATE=CA]
PRODUCT [NAME=Speed Drill Pro, PARTNUM=123XYZ, PLANT=Pittsburgh, INVENTORY=Backordered, CATEGORY=Shop]
SPECIFICATIONS [WEIGHT=8lbs., POWER=120v]
OPTIONS [ADAPTER=Included]
PRICE [MSRP=$149.95, SHIPPING=$15.00]
PRODUCT [NAME=Speed Drill, PARTNUM=124ABC, PLANT=Milwaukee, INVENTORY=InStock, CATEGORY=HandTool]
SPECIFICATIONS [WEIGHT=7.5lbs., POWER=120v]
OPTIONS [ADAPTER=Optional]
PRICE [MSRP=$99.95, SHIPPING=$10.00]
PRODUCT [NAME=Sawzlt, PARTNUM=456XYZ, PLANT=Chicago, INVENTORY=InStock, CATEGORY=Table]
SPECIFICATIONS [WEIGHT=135lbs., POWER=240v]
OPTIONS [ADAPTER=NotApplicable]
PRICE [MSRP=$149.95, SHIPPING=$15.00]

Figure 1

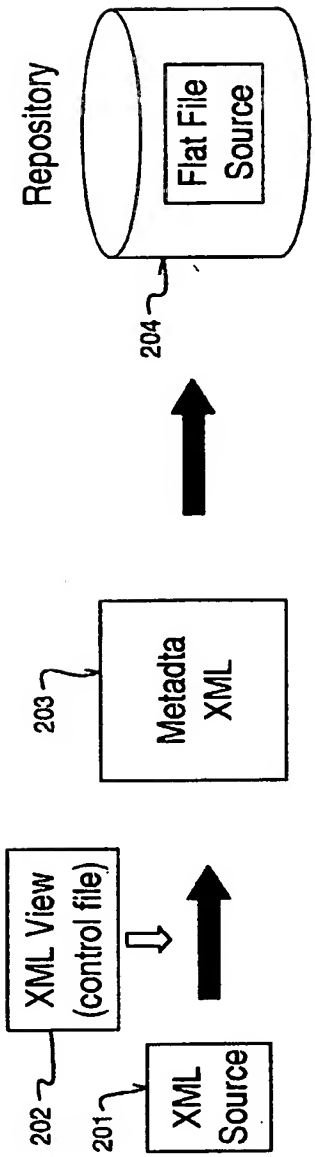

Repository
Flat File Source
204
Metadta XML
203
XML View (control file)
202
XML Source
201

Figure 2

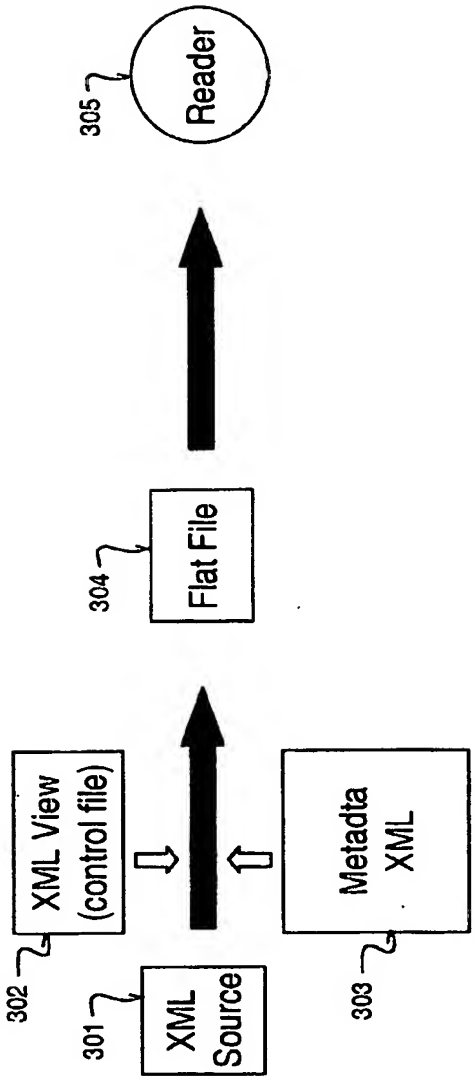

305
Reader
304
Flat File
302
XML View
(control file)
301
XML
Source
303
Metadta
XML

Figure 3

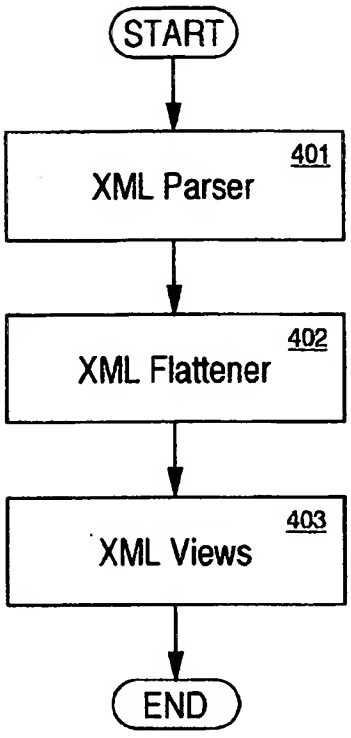


Figure 4

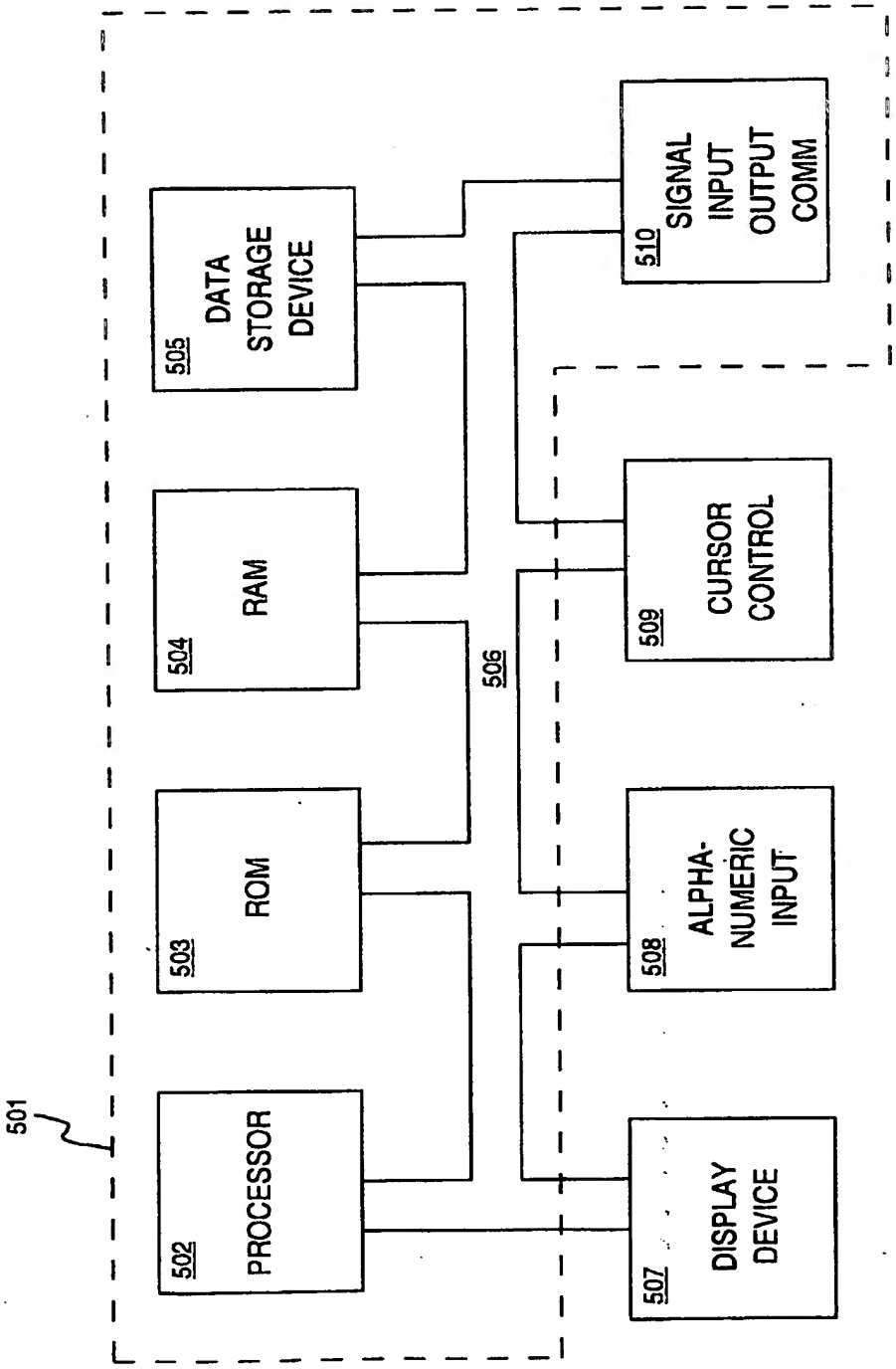

501
PROCESSOR
502
ROM
503
RAM
504
DATA STORAGE DEVICE
505
506
SIGNAL INPUT OUTPUT COMM
510
CURSOR CONTROL
509
ALPHA-NUMERIC INPUT
508
DISPLAY DEVICE
507

Figure 5